UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/684,053	10/09/2003	Chandan Mathur	1934-12-3	3240

7590 04/08/2008

Bryan A. Santarelli
GRAYBEAL JACKSON HALEY LLP
Suite 350
155-108th Avenue NE
Bellevue, WA 98004-5901

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

MAIL DATE	DELIVERY MODE
04/08/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	**Application No.**	**Applicant(s)**	
	10/684,053	MATHUR ET AL.	
	Examiner	**Art Unit**	
	DAVID J. HUISMAN	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *27 December 2007*.

2a)☐ This action is **FINAL**. 2b)☒ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-61* is/are pending in the application.

4a) Of the above claim(s) *25-36 and 55-61* is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-15,17-24 and 37-54* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *14 May 2004* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)

2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)

3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.

4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .

5) ☐ Notice of Informal Patent Application

6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-61 are pending. Claims 25-36 and 55-61 have been withdrawn. Claims 1-15, 17-24, and 37-54 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: RCE, Extension of Time, and Amendment as received on 12/27/2007.

Specification

3. The amended title of the invention submitted by applicant on December 27, 2007, is not descriptive. A new title is required that is clearly indicative of the invention to which the claims are directed.

Claim Objections

4. Claim 48 is objected to because of the following informalities: Please delete either "further comprising:" or "wherein receiving the message comprises". Appropriate correction is required.

5. Claim 53 is objected to because of the following informalities: In line 10, replace “a processor" with --the processor--. Appropriate correction is required.

Claim Rejections - 35 USC § 112

6. The following is a quotation of the first paragraph of 35 U.S.C. 112:

> The specification shall contain a written description of the invention, and of the manner and process of making and using it, in such full, clear, concise, and exact terms as to enable any person skilled in the art to which it pertains, or with which it is most nearly connected, to make and use the same and shall set forth the best mode contemplated by the inventor of carrying out his invention.

7. Claims 10-18 are rejected under 35 U.S.C. 112, first paragraph, as failing to comply with the written description requirement. The claim(s) contains subject matter which was not described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventor(s), at the time the application was filed, had possession of the claimed invention. Specifically, applicant has amended claim 10 such that the processor <u>generates data under the control of the application</u>, retrieves and loads the generated data into the buffer, unloads the data from the buffer, and processes the unloaded data under control of the application. The examiner has been unable to find support in the original disclosure for an application that causes both, under control of the application, the generation of data and the processing of the same data after it is loaded and unloaded. Applicant does have original support for the retrieval of data from a buffer and the processing of the data with an application. See the abstract and the summary of the invention. However, applicant has no original support for the generation of the data under the control of the application. Presumably, this data was generated by another component, and then retrieved by the processor set forth in the abstract. If the examiner is incorrect in his assertion, applicant is respectfully requested to point out support for such a processor in the original disclosure. A further issue exists in claim 18 where claims 10 and 18 together essentially set forth a system in which a single processor generates data under control of an application, packages the generated data into a message that includes the header and data, retrieves the generated data from the application and loads it into a buffer, unloads the data from the buffer, and then processes the unloaded data. The examiner is not clear why the

processor packages data in a message, buffers it, and then unloads it just to continue processing it (the processor appears to be sending a message to itself). The examiner wonders if applicant should be claiming a processor that generates a message that is sent to another processor and not to itself. If applicant believes the examiner to be incorrect in his assertion, applicant is respectfully requested to point out support for a processor which packages data in a message, buffers it, and then unloads it just to continue processing it.

8. Claims 11-18 are rejected under 35 U.S.C 112, 1st paragraph, for containing new matter, because they are dependent, either directly or indirectly, on a claim which includes new matter.

Claim Rejections - 35 USC § 102

9. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

10. Claims 1-3, 5-12, 14-15, 17-18, 37, 39-43, 45, and 47-49 are rejected under 35 U.S.C. 102(b) as being anticipated by Dretzka et al., U.S. Patent No. 4,703,475 (herein referred to as Dretzka). Note that the same claims are rejected multiple times under different interpretations of Dretzka.

11. Referring to claim 1, Dretzka has taught a computing machine, comprising:

a) first and second parallel buffers. See Fig.5, components 120-0 and 120-4, for instance.

b) a processor (Fig.1, component 11) coupled to the buffers and operable to:

b1) execute an application and first, second, third, and fourth data-transfer objects. All processors inherently execute an application. And, Fig.3 sets forth at least some of the data-transfer objects executed by processor 11. Looking at Fig.3 and Fig.5, a first transfer object would be executed to load data into buffer 120-0, a second object would be executed to load data into buffer 120-4, and third object would be executed to retrieve data from buffer 120-0 for loading into buffer 130-0, and a fourth object would be executed to retrieve data from buffer 120-4 for loading into buffer 130-4.

b2) publish data under the control of the application. See the top of Fig.3 and note that the processor produces (publishes) data messages as a result of application execution.

b3) load the published data into the first and second buffers under the control of the first and second data-transfer objects, respectively. See Fig.3 and Fig.5 and note that the published data may be written to any of the message buffers 120-0, 120-4, etc. over time, under control of the associated transfer objects.

b4) retrieve the published data from the first and second buffers under the control of the third and fourth data-transfer objects, respectively. See Fig.3, Fig.5, and column 8, line 66, to column 9, line 4. The published data is retrieved from the first and second buffers under the control of the associated transfer objects.

12. Referring to claim 2, Dretzka has taught the computing machine of claim 1 wherein:

a) the first and third data-transfer objects respectively comprise first and second instances of first object code. See Fig.5 and note that, in general, the first object retrieves data from a previous buffer 110 and stores it in a next buffer 120-0 in channel 0. Likewise, the third object retrieves data from a previous buffer 120-0 and stores it in a next buffer 130-0 in channel 0. Hence, both of these objects comprise instances of general "channel 0 retrieve-and-store" object code.

b) the second and fourth data-transfer objects respectively comprise first and second instances of second object code. See Fig.5 and note that, in general, the second object retrieves data from a previous buffer 110 and stores it in a next buffer 120-4 in channel 4. Likewise, the fourth object retrieves data from a previous buffer 120-4 and stores it in a next buffer 130-4 in channel 4. Hence, both of these objects comprise instances of general "channel 4 retrieve-and-store" object code.

13. Referring to claim 3, Dretzka has taught the computing machine of claim 1 wherein the processor comprises:

a) a processing unit operable to execute the application and publish the data under the control of the application. Recall from the rejection of claim 1 (and from Fig.3) that the processor inherently executes an application and publishes data under control of the application. This execution and publishing is performed by a processing unit.

b) a data-transfer handler operable to execute the first, second, third, and fourth data-transfer objects, to load the published data into the first and second buffers under the control of the first and second data-transfer objects, respectively, and to retrieve the published data from the first and second buffers under the control of the third and fourth data-transfer objects, respectively. Again, recall from the rejection of claim 1 that first and second objects loading the buffers in level 3 of Fig.5, and third and fourth buffers retrieve the data from buffers in level 3 of Fig.5. The objects which perform this are generally taught in Fig.3, where separate objects would clearly exist for each channel as the exact same object could not specify a different channel. The unit which performs this execution for loading and retrieving data is a data transfer handler.

14. Referring to claim 5, Dretzka has taught the computing machine of claim 1 wherein the processor is further operable to:

a) execute a queue object and a reader object. See Fig.3 and Fig.5. The queue object is the object which causes data from buffer 110 to at least be broken up, attached to a header, and have a "more" bit set, before being stored in buffers in level 3. The reader object is the object which at least reads the "more" bit to cause further action to occur.

b) store a queue value under the control of the queue object, the queue value reflecting the loading of the published data into the first buffer. See Fig.3 and Fig.5. Values are written into the level 3 storage until the "more" bit is set low, which means loading of the data is done.

c) read the queue value under the control of the reader object. The "more" bit would not be set for no reason. It clearly serves a purpose, and it will be read. The object which reads it is the reader object.

d) notify the third data-transfer object that the published data occupies the first buffer under the control of the reader object and in response to the queue value. When the "more" bit is set and noted by the reader object, the data may be transferred to the next level of buffers. See Fig.3 and Fig.5.

e) retrieve the published data from the first buffer under the control of the third data-transfer object and in response to the notification. Again, in response to the third object, the data will be moved from level 3 buffers to level 2.5 buffers.

15. Referring to claim 6, Dretzka has taught the computing machine of claim 1, further comprising:

a) a bus. See Fig.2, at least components 40-4, 40-3, etc.

b) wherein the processor is operable to execute a communication object and to drive the data retrieved from one of the first and second buffers onto the bus under the control of the communication object. See Fig.2 and Fig.5. Note that after data is stored in a level 2.5 buffer (second buffers), a communication object will ultimately drive that data onto the bus via a digital facility interface (DFI) 14-0, 14-1, etc.

16. Referring to claim 7, Dretzka has taught the computing machine of claim 1, further comprising:

a) a third buffer. See Fig.5, component 130-0, for instance.

b) wherein the processor is operable to provide the data retrieved from one of the first and second buffers to the third buffer under the control of the respective one of the third and fourth data-transfer objects. Recall from Fig.5 that the first buffer is buffer 120-0 and that data will ultimately be moved from buffer 120-0 to buffer 130-0. The third object is the object which moves the data between these two buffers the first and third buffer.

17. Referring to claim 8, Dretzka has taught the computing machine of claim 1 wherein the processor is further operable to generate a message that includes a header and data retrieved from one of the first and second buffers under the control of the respective one of the third and fourth data-transfer objects. See Fig.3 (at least the level 2.5 object code), and note that the third object removes the data from buffers in level 3 and adds a header to the data to form a 22-byte packet.

18. Referring to claim 9, Dretzka has taught the computing machine of claim 1 wherein:

a) the first and third data-transfer objects respectively comprise first and second instances of first object code. See Fig.5 and note that, in general, the first object retrieves data from a previous buffer 110 and stores it in a next buffer 120-0 in channel 0. Likewise, the third object retrieves data from a previous buffer 120-0 and stores it in a next buffer 130-0 in channel 0. Hence, both of these objects comprise instances of general "channel 0 retrieve-and-store" object code.

b) the second and fourth data-transfer objects respectively comprise first and second instances of second object code. See Fig.5 and note that, in general, the second object retrieves data from a previous buffer 110 and stores it in a next buffer 120-4 in channel 4. Likewise, the fourth object retrieves data from a previous buffer 120-4 and stores it in a next buffer 130-4 in channel 4. Hence, both of these objects comprise instances of general "channel 4 retrieve-and-store" object code.

c) the processor is operable to execute an object factory and to generate the first object code and the second object code under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for data to be transmitted, the first and second object codes will be generated and invoked so that data may be transmitted.

19. Referring to claim 10, Dretzka has taught a computing machine, comprising:

a) a first buffer. See Fig.6, buffer 220-4, for instance.

b) a processor (Fig.1, component 21) coupled to the buffer and operable to:

b1) execute first and second data-transfer objects and an application. Fig.4 sets forth at least some of the data-transfer objects executed by processor 21. Looking at Fig.4 and Fig.6, a first transfer object would be executed to load data into buffer 220-4 and a second object would be executed to unload data from buffer 220-4. At least some of the other functions performed by the processor (for instance, doing normal ALU operations) would be part of the application inherently executed by the processor.

b2) generate data under control of the application. See Fig.6 and Figs.8-15, and note that the application generates data using an input list.

b3) retrieve the generated data from the application and load the retrieved data into the buffer under the control of the first data-transfer object. See Fig.6. After data is generated by the input list, the first object transfers it to the buffer (i.e., 220-4).

b3) unload the data from the buffer under the control of the second data-transfer object. See Fig.6. Data is ultimately moved/unloaded from the buffer 220-4 and into buffer 210 under control of the second object.

b4) process the unloaded data under the control of the application. See Fig.4 and Fig.6. From the buffer 210, the processor will process the data using an application.

20. Referring to claim 11, Dretzka has taught the computing machine of claim 10 wherein the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.6 and note that, in general, the first object retrieves data from a previous stage (level 2.5 stage) and stores it in a next buffer 220-4. Likewise, the second object retrieves data from a previous stage (level 3) and stores it in a next buffer 210. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

21. Referring to claim 12, Dretzka has taught the computing machine of claim 10 wherein the processor further comprises:

a) a processing unit operable to execute the application, generate the data, and process the unloaded data under the control of the application. Recall from the rejection of claim 10 (and from Fig.4) that the processor performs each of these claimed functions. They are inherently performed by a processing unit.

b) a data-transfer handler operable to execute the first and second data-transfer objects, to retrieve the data from the application and load the data into the buffer under the control of the first data-transfer object, and to unload the data from the buffer under the control of the second data-transfer object. Again, recall from the rejection of claim 10 that the claimed objects are executed. The unit which performs this execution is a data transfer handler.

22. Referring to claim 14, Dretzka has taught the computing machine of claim 10 wherein the processor is further operable to:

a) execute a queue object and a reader object. See Fig.4 and Fig.6. The queue object is the object which stores the more bit, which ultimately leads to the informing of level 4 that a complete message has been received. The reader object is the object which detects the more bit so that further action can occur.

b) store a queue value under the control of the queue object, the queue value reflecting the loading of the retrieved data into the first buffer. See Fig.4 and Fig.6. The queue object will store the "more" bit, which reflects the loading of the retrieved data into the buffer. This bit, when set in a certain manner, will allow for the informing of level 4 of a complete message.

c) read the queue value under the control of the reader object. See Fig.4 and note that the more bit, when set to a certain level, will indicate the end of the message and that the message may be further transmitted and processed. The reader object will have to read this more bit.

d) notify the second data-transfer object that the retrieved data occupies the buffer under the control of the reader object and in response to the queue value. When the “more” bit is set in the appropriate manner and noted by the reader object, the data may be transferred to the next-level buffer. See Fig.4 and Fig.6.

e) unload the retrieved data from the buffer under the control of the second data-transfer object and in response to the notification. Again, in response to the notification, the data will be moved from level 3 buffer to level 4 buffer. See Fig.4 and Fig.6.

23. Referring to claim 15, Dretzka has taught the computing machine of claim 10, further comprising:

a) a second buffer. See Fig.6, component 210.

b) wherein the processor is operable to execute a third data-transfer object, to unload the data from the first buffer into the second buffer under the control of the second data-transfer object, and to provide the data from the second buffer to the application under the control of the third data-transfer object. See Fig.4 and Fig.6. Data is unloaded from buffer 220-4 to buffer 210 under control of the second transfer object, and then moved from buffer 210 to the application in the processor under control of the third object.

24. Referring to claim 17, Dretzka has taught the computing machine of claim 10 wherein:

a) the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.6 and note that, in general, the first object retrieves data from a previous stage (level 2.5 stage) and stores it in a next buffer 220-4. Likewise, the second object retrieves data from a previous stage (level 3) and stores it in a next buffer 210. Hence, both of these objects comprise instances of general “retrieve-and-store” object code.

b) the processor is operable to execute an object factory and to generate the object code under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for data to be received, the first and second object codes will be generated and invoked so that data may be received.

25. Referring to claim 10, Dretzka has taught a computing machine (under a second interpretation), comprising:

a) a first buffer. See Fig.5, buffer 120-4, for instance.

b) a processor (Fig.1, component 11) coupled to the buffer and operable to:

> b1) execute first and second data-transfer objects and an application. Fig.3 sets forth at least some of the data-transfer objects executed by processor 11. Looking at Fig.3 and Fig.5, a first transfer object would be executed to load data into buffer 120-4 and a second object would be executed to unload data from buffer 120-4 and into buffer 130-4. At least some of the other functions performed by the processor (for instance, doing normal ALU operations) would be part of the application inherently executed by the processor.
>
> b2) generate data under control of the application. See the top of Fig.3 and note that the processor generates data as a result of application execution.
>
> b3) retrieve the generated data from the application and load the retrieved data into the buffer under the control of the first data-transfer object. See Fig.5. After data is generated, it is ultimately stored in buffer 120-4 (in the example shown).

b3) unload the data from the buffer under the control of the second data-transfer object. See Fig.5. Data is ultimately moved/unloaded from the buffer 120-4 and into buffer 130-4 under control of the second object.

b4) process the unloaded data under the control of the application. See Fig.3 and note that the data, after being removed from buffer 120-4, is further processed by adding a 1-byte header to the data and then assigning the data to a physical link for transmission.

26. Referring to claim 11, Dretzka has taught the computing machine of claim 10 (under a second interpretation) wherein the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.5 and note that, in general, the first object retrieves data from a previous stage (level 4 stage) and stores it in a next buffer 120-4. Likewise, the second object retrieves data from a previous stage (level 3) and stores it in a next buffer 130-4. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

27. Referring to claim 12, Dretzka has taught the computing machine of claim 10 (under a second interpretation) wherein the processor further comprises:

a) a processing unit operable to execute the application, generate the data, and process the unloaded data under the control of the application. Recall from the rejection of claim 10 (and from Fig.3) that the processor performs each of these claimed functions. They are inherently performed by a processing unit.

b) a data-transfer handler operable to execute the first and second data-transfer objects, to retrieve the data from the application and load the data into the buffer under the control of the first data-transfer object, and to unload the data from the buffer under the control of the second data-transfer object. Again, recall from the rejection of claim 10 that the claimed objects are executed. The unit which performs this execution is a data transfer handler.

28. Referring to claim 14, Dretzka has taught the computing machine of claim 10 (under a second interpretation) wherein the processor is further operable to:

a) execute a queue object and a reader object. See Fig.3 and Fig.5. The queue object is the object which causes a "more" bit to be set for data from buffer 110. The reader object is the object which at least reads the "more" bit to cause further action to occur.

b) store a queue value under the control of the queue object, the queue value reflecting the loading of the retrieved data into the first buffer. See Fig.3 and Fig.5. Values (both data and "more" bit) are written into the level 3 storage until the "more" bit is set low, which means loading of the data is done.

c) read the queue value under the control of the reader object. The "more" bit would not be set for no reason. It clearly serves a purpose, and it will be read. The object which reads it is the reader object.

d) notify the second data-transfer object that the retrieved data occupies the buffer under the control of the reader object and in response to the queue value. When the "more" bit is set and noted by the reader object, the data may be transferred to the next level of buffers. See Fig.3 and Fig.5.

e) unload the retrieved data from the buffer under the control of the second data-transfer object and in response to the notification. Again, in response to the notification, the data will be moved from buffer 120-4 to buffer 130-4.

29. Referring to claim 15, Dretzka has taught the computing machine of claim 10 (under a second interpretation), further comprising:

a) a second buffer. See Fig.5, component 130-4.

b) wherein the processor is operable to execute a third data-transfer object, to unload the data from the first buffer into the second buffer under the control of the second data-transfer object, and to provide the data from the second buffer to the application under the control of the third data-transfer object. See Fig.3 and Fig.5. Data is unloaded from buffer 120-4 to buffer 130-4 (second buffer) under control of the second transfer object, and then moved from buffer 130-4 to the interface under control of the third object.

30. Referring to claim 17, Dretzka has taught the computing machine of claim 10 (under a second interpretation) wherein:

a) the first and second data-transfer objects respectively comprise first and second instances of the same object code. See Fig.5 and note that, in general, the first object retrieves data from a previous stage (level 4 stage) and stores it in a next buffer 120-4. Likewise, the second object retrieves data from a previous stage (level 3) and stores it in a next buffer 130-4. Hence, both of these objects comprise instances of general "retrieve-and-store" object code.

b) the processor is operable to execute an object factory and to generate the object code under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for data to be transmitted, the first and second object codes will be generated and invoked so that data may be transmitted.

31. Referring to claim 18, Dretzka has taught the computing machine of claim 10 (under a second interpretation) wherein the processor is further operable to package the generated data into a message that includes a header and the data under the control of the second data-transfer object. See Fig.3 (at least the level 2.5 object code), and note that the second object adds a header to the generated data to form a 22-byte packet.

32. Referring to claim 37, Dretzka has taught a method comprising:

a) publishing with an application data that includes no information indicating a destination of the data. See the top of Fig.3 and note that the processor produces (publishes) data messages as a result of application execution. Note that, at the time of generation, no destination information is produced.

b) loading the published data into a first buffer with a first data-transfer object. See Fig.3 and Fig.5 and note that the published data may be written to buffer 120-4 under control of the first transfer object (the code which performs at least the loading).

c) retrieving the published data from the buffer with a second data-transfer object. See Fig.3, Fig.5, and column 8, line 66, to column 9, line 4. The published data is retrieved from the first buffer under the control of the second transfer object (the code which performs at least the retrieving).

d) generating a message header that includes a destination of the retrieved data. See Fig.3 and note the code performed at level 2.5. In this step, a message header including logical channel number destination is generated and attached to the data.

e) generating a message that includes the retrieved data and the message header. See Fig.3 and note that a 22-byte message is produced from the 21-byte data and 1-byte header.

33. Referring to claim 39, Dretzka has taught the method of claim 37, further comprising:

a) generating a queue value that corresponds to the presence of the published data in the buffer. See Fig.3 and note that when the last part of the message is found, it is indicated by generating a "more" bit (queue value).

b) notifying the second data-transfer object that the published data occupies the buffer in response to the queue value. Note from Fig.3 that when the last packet is found and the more bit is set, the code which then retrieves the data must be notified.

c) wherein retrieving the published data comprises retrieving the published data from the buffer with the second data-transfer object in response to the notification. See Fig.3, and note the level 2.5 code.

34. Referring to claim 40, Dretzka has taught the method of claim 37, further comprising driving the message onto a bus with a communication object. See Fig.3 and Fig.5. After being stored in the level 2.5 buffer, the message is driven on the bus with a communication object.

35. Referring to claim 41, Dretzka has taught the method of claim 37, further comprising loading the retrieved data into a second buffer with the second data-transfer object. See Fig.3 and Fig.5 and note that after being stored in a first buffer 120-4, the second object retrieves the data and stores it in another buffer 130-4, for instance.

36. Referring to claim 42, Dretzka has taught the method of claim 37 wherein generating the message header and the message comprise generating the message header and the message with

the second data transfer object. See Fig.3, level 2.5 code, in which the header is generated and attached to the data.

37. Referring to claim 43, Dretzka has taught the method of claim 37, further comprising:

a) generating data-transfer object code with an object factory. All processors execute programs. The program (object factory) will dictate/generate when data needs to be transmitted and received. That is, when the program calls for data to be transmitted or received, the data transfer objects will be invoked so that data may be transmitted and received. Clearly, the hardware shown in the figures must have some software interaction because without software, the hardware would be useless.

b) generating the first data-transfer object as a first instance of the object code. When data needs to be transferred, some type of "send" or "pass" or "store" command must be generated. This command is part of the data transfer object.

c) generating the second data-transfer object as a second instance of the object code. Similarly, when data needs to be transferred from first buffer to second buffer for header and message generation, some type of command needs to be generated. This command is part of the second data transfer object.

38. Referring to claim 45, Dretzka has taught a method comprising:

a) receiving a message that includes data and that includes a message header that indicates a destination of the data. See Fig.4 and Fig.6. Note that a message comes in with a 1-byte header (note level 2.5).

b) loading into a first buffer with a first data-transfer object, the received data without the message header, the first buffer corresponding to the destination. See Fig.6 and note that the data is loaded into buffer 220-4 without the 1-byte message header (the 1-byte header is removed prior to storage in 220-4 according to Fig.4). The first buffer is located in a channel specified by the destination information sent with the message (see Fig.3).

c) unloading the data from the buffer with a second data-transfer object. See Fig.4 and Fig.6 and note that data is ultimately unloaded from buffer 220-4.

d) processing the unloaded data with an application corresponding to the destination. See Fig.4 and note that after the data is unloaded, it will be sent to the processor where inherent processing on that data will commence.

39. Referring to claim 47, Dretzka has taught the method of claim 45, further comprising:

a) generating a queue value that corresponds to the presence of the data in the buffer. See Fig.4 and Fig.6. The queue object will generates the "more" bit, which corresponds to the presence of data in the buffer. This bit, when set in a certain manner, will allow for the informing of level 4 of a complete message.

b) notifying the second data-transfer object that the data occupies the buffer in response to the queue value. When the "more" bit is set in the appropriate manner and noted by the reader object, the data may be transferred to the next-level buffer by informing (notifying) the next level that data is ready to be transferred. See Fig.4 and Fig.6.

c) wherein unloading the data comprises unloading the data from the buffer with the first data-transfer object in response to the notification. Again, in response to the notification, the data will be moved from level 3 buffer to level 4 buffer. See Fig.4 and Fig.6.

40. Referring to claim 48, Dretzka has taught the method of claim 45, further comprising wherein receiving the message comprises receiving the message with the first data-transfer

object. See Fig.4 and Fig.6. Some code/object will cause the receiving of the data. That object is the first data transfer object.

41. Referring to claim 49, has taught the method of claim 45, further comprising:

a) receiving the message comprises retrieving the message from a bus with a communication object. See Fig.4 and Fig.6. Some code/object will cause the receiving of the data from a bus. That object is the communication object.

b) transferring the data from the communication object to the first data transfer object. See Fig.4 and Fig.6. Note that after the data is received, it is pass to the first data transfer object which at least stores it in buffer 220-4.

Claim Rejections - 35 USC § 103

42. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

43. Claims 4, 13, 38, 44, 46, and 50 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dretzka in view of the examiner's taking of Official Notice.

44. Referring to claim 4, Dretzka has taught the computing machine of claim 1. Dretzka has not taught that the processor is further operable to execute a thread of the application and to publish the data under the control of the thread. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by

reducing stall time. With multiple threads, the system may switch to a second thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor executes a thread of the application and publishes data under control of the thread.

45. Referring to claim 13, Dretzka has taught the computing machine of claim 10 (under both the first and second interpretations of Dretzka). Dretzka has not taught that the processor is further operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, since threads are independent sequences of instructions, the system may switch to a next thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, and because generating data and processing unloaded data are independent tasks, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor is operable to execute first and second threads of the application, to generate the data under the control of the first thread, and to process the unloaded data under the control of the second thread.

46. Referring to claim 38, Dretzka has taught the method of claim 37. Dretzka has not taught that publishing the data comprises publishing the data with a thread of the application. However, Official Notice is taken that multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, since threads are independent sequences of instructions, the system may switch to a next thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that publishing the data comprises publishing the data with a thread of the application.

47. Referring to claim 44, Dretzka has taught the method of claim 37. Dretzka has further taught receiving the message and processing the data in the message with a processor (Fig.4 and Fig.6). Dretzka has not explicitly taught the receiving processor is a pipeline accelerator. However, Official Notice is taken that pipelined processors and their advantages are well known and accepted in the art. A pipeline allows for the overlapping of execution, instead of serial execution, thereby speeding up the processor and increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka's processor (Fig.1, component 21) such that it includes a pipeline for accelerating execution.

48. Referring to claim 46, Dretzka has taught the method of claim 45. Dretzka has not taught that processing the unloaded data comprises processing the unloaded data with a thread of the application corresponding to the destination. However, Official Notice is taken that

multithreaded processors and their advantages are well known and accepted in the art. Specifically, it is known to divide up a program into threads in order to increase efficiency by reducing stall time. With multiple threads, the system may switch to a second thread when a first thread stalls, thereby hiding the stall time require by the first thread. Essentially, the processor is kept busy as often as possible with multithreading. As a result, in order to increase efficiency, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka such that the processor processes the unloaded data with a thread.

49. Referring to claim 50, Dretzka has taught the method of claim 45. Dretzka has not explicitly taught generating the message header and the message with a pipeline accelerator. However, Official Notice is taken that pipelined processors and their advantages are well known and accepted in the art. A pipeline allows for the overlapping of execution, instead of serial execution, thereby speeding up the processor and increasing throughput. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka's processor (Fig.1, component 11) such that it includes a pipeline for accelerating execution. Note that in this series of rejections, processor 11 is the unit which generates the messages according to Fig.3.

50. Claims 19-24 and 51-54 are rejected under 35 U.S.C. 103(a) as being unpatentable over Dretzka in view of Britton et al., U.S. Patent No. 6,216,191 (herein referred to as Britton).

51. Referring to claim 19, Dretzka has taught a peer-vector machine comprising:

a) a buffer. See Fig.5, component 120-4.

b) a bus. See Fig.2, components 40-0, 40-1, etc.

c) a processor (Fig.1, components 11) coupled to the buffer and to the bus and operable to:

c1) execute an application, first and second data-transfer objects, and a communication object. And, Fig.3 sets forth at least some of the data-transfer objects executed by processor 11. Looking at Fig.3 and Fig.5, a first transfer object would be executed to load data into buffer 120-0, a second object would be executed to load data into buffer 120-4, and third object would be executed to retrieve data from buffer 120-0 for loading into buffer 130-0, and a fourth object would be executed to retrieve data from buffer 120-4 for loading into buffer 130-4. At least some functions, other than those performed by the first and second objects, are performed by the communication object.

c2) publish data under the control of the application. See the top of Fig.3 and note that the processor produces (publishes) data messages as a result of application execution.

c3) load the published data into the buffer under the control of the first data-transfer object. See Fig.3 and Fig.5 and note that the published data may be written to message buffers 120-4, etc. over time, under control of the first transfer object.

c4) retrieve the published data from the buffer under the control of the second data-transfer object. See Fig.3, Fig.5, and column 8, line 66, to column 9, line 4. The published data is retrieved from the first buffer under the control of the second transfer object.

c5) construct a message under the control of the second data-transfer object, the message including the retrieved published data and information indicating a destination of the retrieved published data. See Fig.3 and note that a 1-byte header including destination information (channel number) is attached to the data, thereby forming a message to send to the receiver.

c6) drive the published data onto the bus under the control of the communication object. See Fig.2 and Fig.5. Note that after data is stored in a level 2.5 buffer (second buffer), a communication object will ultimately drive that data onto the bus via a digital facility interface (DFI) 14-0, 14-1, etc.

d) a processor coupled to the bus, including the destination, and operable to receive the message from the bus, to recover the received published data from the message, to provide the recovered data to the destination, and to process the recovered data at the destination. See Fig.1 and Figs.3-4. Note that the destination would specify "the other processor through channel LCN#" The circuitry of Fig.6 would receive the message , extract the data, and then send it to the receiving processor, where it will be processed.

Dretzka has not taught that the processor coupled to the bus is a pipeline accelerator coupled to the bus and that the recovered data is processed at the destination without executing a program instruction. However, Britton has taught the general concept of coupling a processor and an FPGA. See Fig.1. Consequently, the processor could communicate with the FPGA and access its user-defined circuitry, thereby accessing custom hardware for performing operations. Custom hardware is simply reactionary to data. No instruction is executed. As a result, it order to access custom circuitry for operation, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka's processor 21 to be an FPGA. Furthermore, since pipelining is well known to be advantageous in the art, it would have been obvious to pipeline one or more of the processor and FPGA to increase throughput.

52. Referring to claim 20, Dretzka in view of Britton has taught the peer vector machine of claim 19 wherein the destination includes a field-programmable gate array that is operable to process the recovered data. See Britton, Fig.1, component 104.

53. Referring to claim 21, Dretzka in view of Britton has taught the peer vector machine of claim 19, further comprising

a) a registry coupled to the processor and operable to store object data. The examiner asserts that the processor's objects are made up of instructions which must be stored somewhere so that the processor may access and execute them. The storage holding the instructions is the registry.

b) wherein the processor is operable to execute an object factory, and to generate the first and second data-transfer objects and the communication object from the object data under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for data to be transmitted, the first and second object codes will be generated and invoked so that data may be transmitted.

54. Referring to claim 22, Dretzka has taught a peer vector machine comprising:

a) a buffer. See Fig.6, component 220-4, for instance.

b) a bus. See Fig.2, component 40-1, 40-2, etc.

c) a unit coupled to the bus and operable to generate data, to generate a header including information indicating a destination of the data, to package the data and header into a message, and to drive the message onto the bus. See Fig.1, unit 11, and Fig.3 and Fig.5. Note that the unit generates data, and then a message including the data and a header (with destination data specifying "other processor via channel LCN#"), and then drives that data onto bud 40-0, 40-1, etc (Fig.2), through the digital facility interface.

d) a processor (Fig.1, component 21) coupled to the buffer and to the bus and operable to:

d1) execute an application, first and second data-transfer objects, and a communication object. Fig.4 sets forth at least some of the data-transfer objects executed by processor 21. Looking at Fig.4 and Fig.6, a first transfer object would be executed to load data into buffer 220-4 and a second object would be executed to unload data from buffer 220-4 and into buffer 210. The communication object would be executed to retrieve data from the DFI 14-0, 14-1, etc, and store it into one of buffers 230-4. At least some of the other functions performed by the processor (for instance, doing normal ALU operations) would be part of the application inherently executed by the processor.

d2) receive the message from the bus under the control of the communication object. See Fig.4 and Fig.6. Note that a message is received from the interface under the control of the communication object.

d3) load into the buffer under the control of the first data-transfer object the received data without the header, the buffer corresponding to the destination of the data. See Fig.4 and Fig.6 and note that before being stored in the first buffer in level 3, the 1-byte header is deleted.

d4) unload the data from the buffer under the control of the second data-transfer object. See Fig.4 and Fig.6 and note that from the first buffer, the data is moved to the buffer 210 before ultimately being sent to the processor 21.

d5) process the unloaded data under the control of the application. See Fig.4 and Fig.6. From the buffer 210, the processor will process the data using an application.

e) Dretzka has not taught that the <u>unit</u> coupled to the bus is a pipeline accelerator coupled to the bus and that the recovered data is processed at the destination without executing a program instruction. However, Britton has taught the general concept of coupling a processor and an FPGA. See Fig.1 and note the bidirectional bus, meaning both the processor and FPGA can communicate data to each other. Consequently, the processor could communicate with the FPGA and access its user-defined circuitry, thereby accessing custom hardware for performing operations. Custom hardware is simply reactionary to data. No instruction is executed. As a result, it order to access custom circuitry for operation, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka's unit 11 to be an FPGA. Furthermore, since pipelining is well known to be advantageous in the art, it would have been obvious to pipeline one or more of the processor and FPGA to increase throughput.

55. Referring to claim 23, Dretzka in view of Britton has taught the peer vector machine of claim 22 wherein the processor is operable to receive the message from the bus under the control of the communication object, and to recover the data from the message under the control of the first data-transfer object. See Fig.4 and note that the communication object receives the message from the bus. And, the first data object recovers the data from the message. See at least the level 3 code of Fig.4.

56. Referring to claim 24, Dretzka in view of Britton has taught the peer vector machine of claim 22, further comprising:

a) a registry coupled to the processor and operable to store object data. The examiner asserts that the processor's objects are made up of instructions which must be stored somewhere so that the processor may access and execute them. The storage holding the instructions is the registry.

b) wherein the processor is operable to execute an object factory, and to generate the first and second data-transfer objects and the communication object from the object data under the control of the object factory. All processors execute programs. The program (object factory) will dictate when data needs to be transmitted and received. That is, when the program calls for data to be received, the first and second object codes (and communication object codes) will be generated and invoked so that data may be received.

57. Referring to claim 51, Dretzka has taught a method comprising:

a) publishing data with an application running on a processor. See the top of Fig.3 and note that the processor produces (publishes) data as a result of application execution.

b) loading the published data into a buffer with a first data-transfer object running on the processor. See Fig.3 and Fig.5 and note that the published data may be written to buffer 120-4 under control of the first transfer object (the code which performs at least the loading).

c) retrieving the published data from the buffer with a second data-transfer object running on the processor. See Fig.3, Fig.5, and column 8, line 66, to column 9, line 4. The published data is retrieved from the first buffer under the control of the second transfer object (the code which performs at least the retrieving).

d) generating information that indicates a destination of the retrieved data. See Fig.3 and note the code performed at level 2.5. In this step, a message header including logical channel number destination is generated and attached to the data.

e) packaging the retrieved data and the information into a message. See Fig.3 and note that a 22-byte message is produced from the 21-byte data and 1-byte header.

f) driving the message onto a bus with a communication object running on the processor. See Fig.3 and Fig.5. After being stored in the level 2.5 buffer, the message is driven on the bus with a communication object.

g) receiving the message from the bus and processing the published data with a unit. See Fig.4 and Fig.6. Note that unit 20 of Fig.1 receives the messages sent by unit 10.

h) Dretzka has not taught that the unit is a pipeline accelerator that includes a field-programmable gate array. However, Britton has taught the general concept of coupling a processor and an FPGA. See Fig.1 and note the bidirectional bus, meaning both the processor and FPGA can communicate data to each other. Consequently, the processor could communicate with the FPGA and access its user-defined circuitry, thereby accessing custom hardware for performing operations. Custom hardware is simply reactionary to data. No instruction is executed. As a result, it order to access custom circuitry for operation, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka's unit 20 (or at least 21) to be an FPGA. Furthermore, since pipelining is well known to be advantageous in the art, it would have been obvious to pipeline one or more of the processor and FPGA to increase throughput.

58. Referring to claim 52, Dretzka in view of Britton has taught a method as described in claim 51, further comprising:

a) generating a message that includes a header and the published data with the second data-transfer object. See Fig.3 and note that in addition to receiving the data from the level 3 buffer, the second object also generates the header in the level 2.5 code of Fig.4.

b) driving the data onto the bus comprises driving the message onto the bus with the communication object. See Fig.3 and Fig.5. After being stored in the level 2.5 buffer, the message is driven on the bus with a communication object.

c) receiving and processing the published data comprises receiving the message and recovering the published data from the message with the pipeline accelerator. Again, see Fig.4 and Fig.6, and recall the modification in view of Britton.

59. Referring to claim 53, Dretzka has taught a method comprising:

a) generating with a <u>unit</u> a message header that includes a destination of data. See Fig.3 and note the level 2.5 object which generates a 1-byte header including a logical channel destination.

b) generating with <u>the unit</u> a message that includes a header and the data. See Fig.3 and note that in generating the 1-byte header, the unit forms a 22-byte message with 21-bytes of data.

c) driving the message onto a bus with <u>the unit</u>. See Fig.3 and Fig.5. Note that after the messages are generated, they are sent over the bus shown in Fig.2 using the digital facility interface (DFI).

d) receiving the message from the bus with a communication object running on a processor. See Fig.4 and Fig.6. note that processor 21 of Fig.1 is receiving the message using code shown in Fig.4.

e) loading into a buffer with a first data-transfer object running on the processor the received data absent the header, the buffer being identified by the destination. See Fig.4 and note the level 3 object which stores data into the buffer in level 3 minus the header which was stripped in level 2.5 of Fig.4.

f) unloading the data from the buffer with a second data-transfer object running on the processor. See Fig.4 and Fig.6 and note that data from a buffer in level 3 is ultimately removed and moved on through the system.

g) processing the unloaded data with an application running on the processor and identified by the destination. See Fig.4 and Fig.6. Eventually, the unloaded data makes its way to the processor for inherent processing.

f) Dretzka has not taught that the unit is a pipeline accelerator. However, Britton has taught the general concept of coupling a processor and an FPGA. See Fig.1 and note the bidirectional bus, meaning both the processor and FPGA can communicate data to each other. Consequently, the processor could communicate with the FPGA and access its user-defined circuitry, thereby accessing custom hardware for performing operations. Custom hardware is simply reactionary to data. No instruction is executed. As a result, it order to access custom circuitry for operation, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Dretzka's unit 10 (or at least 11) to be an FPGA. Furthermore, since pipelining is well known to be advantageous in the art, it would have been obvious to pipeline one or more of the processor and FPGA to increase throughput.

60. Referring to claim 54, Dretzka in view of Britton has taught a method as described in claim 53, further comprising recovering the data from the message with the first data transfer object. See Fig.4, and note that recovering the data from the message begins with the first data transfer object executing in level 2.5 of Fig.4.

Response to Arguments

61. Applicant's arguments set forth in the remarks filed on December 27, 2007, have been considered but are moot in view of the new ground(s) of rejection.

Conclusion

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DAVID J. HUISMAN whose telephone number is (571)272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/David J. Huisman/

Primary Examiner, Art Unit 2183
March 21, 2008